# Monitoring distributed collections using the Audit Control Environment (ACE)

Michael Smorul
Institute for Advanced Computer Studies
University of Maryland, College Park
+1-301-405-7092
msmorul@umd.edu

Sangchul Song
Department of Electrical and Computer Engineering
Institute for Advanced Computer Studies
University of Maryland, College Park
+1-301-405-7092
scsong@umiacs.umd.edu

Joseph JaJa
Department of Electrical and Computer Engineering
Institute for Advanced Computer Studies
University of Maryland, College Park
+1-301-405-6722
joseph@umiacs.umd.edu

## ABSTRACT
The Audit Control Environment (ACE) is a system which provides a scalable, auditable platform that actively monitors collections to ensure their integrity over the lifetime of an archive. It accomplishes this by using a small integrity token issued for each monitored item. This token is part of a larger externally auditable cryptographic system. We will describe how this system has been implemented for a set of applications designed to run in an archive or library environment.

ACE has been used for almost two years by the Chronopolis Preservation Environment to monitor the integrity of collections replicated between the three independant archive partners. During this time, ACE has been expanded to better support the requirements of this distributed archive. We will describe how ACE has been used and expanded to support the Chronopolis preservation requirements. We conclude by discussing several future requirements for integrity monitoring that have been identified by users of ACE. These include securely monitoring remote data, monitoring offline data, and scaling monitoring activities in a way that does not impact the normal operational activity of an archive.

## Categories and Subject Descriptors
H.3.4 [**Information Storage and Retrieval**]: Systems and Software; H.3.7 [**Information Storage and Retrieval**]: Digital Libraries

## General Terms
Reliability, Verification, Security

## Keywords
Digital Preservation

## 1. INTRODUCTION
The ACE[1] system is a set of tools for digital archives and libraries to monitor and assert the integrity of their collections. ACE serves two primary functions; first it provides an easy-to-use interface for archives to monitor and manage their collections. Second, the integrity information stored in ACE can be independently verified by a third party to ensure that objects in a collection have not been modified. While archives have been using cryptographic digests to verify the integrity of their data, little has been done to ensure those digests have not been tampered with. ACE solves this problem by storing a small integrity token that can be used to externally verify that a digest has not changed.

## 2. DESIGN
ACE is an integrity monitoring platform based on creating a small-size integrity token for each digital object registered into ACE. This token can be stored either with the object itself or in a registry at the archive as authenticity metadata.

These tokens are linked together through time spans by an auditable third party. For each time interval, cryptographic summary information (CSI) that depends on all the objects registered during that time interval is generated. The summary information is very compact, independent of the number or sizes of the objects ingested. In the current implementation, rounds are created after 5 seconds have elapsed since the first token request was received, or a queue of pending requests is full. While triggering a round on queue size may lead to many more rounds in a day, it also limits the size of returned tokens.

At the end of each day, all CSI's generated are aggregated into a final witness value. This witness value is a single number that is used to verify all CSI's issued during the previous day. The value is expected to be stored in reliable, read-only media, and pubished over the internet. Currently, several sources are available on the internet that receive and archive generated witness values. An independent auditor, given a trusted witness, may assert the integrity of all CSIs for a given time period. Once CSIs are certified, they may be used to validate all tokens covered by the summaries. Once tokens are validated, an auditor may assert that any file whose cryptographic digest matches its token has not been tampered with an extremely high probability.

Regular audits will be continuously conducted, which will make use of the integrity tokens and the summary integrity information to ensure the integrity of both the objects and the integrity information. In our implementation, audits can also be triggered by an archive manager or by a user upon data access. However we are assuming that the auditing services are not allowed to change the content of the archive even if errors are detected. The responsibility for correcting errors is left to the archive administrator after being alerted by the auditing service.

The ACE system consists of two components, first is an Integrity Management Service(IMS) which gathers token requests into rounds and generates Integrity Tokens(IT) at the end of each round. The IMS then stores the CSI for that round in its database. The IMS is also responsible for publishing nightly witness values. The University of Maryland currently hosts a publicly available IMS for any party to use. The second component of ACE consists of multiple, independent Audit Managers(AM) that are installed locally at different nodes of the archive and that periodically check the integrity of monitored objects according to a locally defined policy. There are currently two different Audit Managers available, a standalone client and a web based application for monitoring multiple, large collections.

# 3. ACE AUDITS

There are several different types of audits that can be performed within ACE. These are local audits of holdings, challenge from the archive of the IMS integrity, and finally a fully external audit of both the IMS and AM components.

## 3.1 Local Auditing

A local ACE audit consists of an AM validating the data under its domain. For each monitored object, the local audit is performed as follows:

1. The AM computes the digest of the object and retrieves its integrity token.
2. Making use of the stored hash value and the proof in the integrity token, AM computes the round hash value.
3. Using the round id stored in the integrity token, the AM requests the round hash value from the IMS.
4. The audit manager asserts an object is correct if the objects computed hash in Step 1 is equal to the hash value stored in its database and the round hash values computed in steps 2 and 3 are equal. Otherwise an error is logged specifying which step failed.

It is clear that if the two object hash values, as well as the two round hash values are equal, then the object and integrity token are intact with a very high probability.

## 3.2 IMS Challenge

The next type of validation that will occur is the challenging of the IMS by the AM or a third party to determine if its round hashes are correct. The process for challenging the IMS is described below:

1. Extract the round ID and daily witness value for that round from the integrity token and local witness database.
2. Request the IMS supply a proof for the requested round.
3. Compute a witness value using the round hash agreed upon in local auditing and the IMS supplied proof.
4. The IMS is considered intact if the computed witness value and the stored witness value match.

This auditing process allows for the audit manager to store a minimal amount of data, the witness, and prove that the IMS is correctly supplying round summaries. To further secure the process, the witness values should be kept near the archive on separate, trusted storage. This only allows an archive to assert that it is internally consistent, we will describe below how a third party is able to audit an archive.

## 3.3 Full External Audit

An archive must be able to prove that its holdings are valid. When an archive is audited by a 3rd party to ensure that its holdings are intact it may present its stored objects along with their integrity tokens to the 3rd party.

1. The 3rd party securely obtains a copy of any applicable witness values by both subscribing to the witness broadcast and storing them locally, or by using a trusted source. The trusted source should be independent from the archive being audited.
2. The IMS performs the steps described in 3.1.
3. If IMS integrity is to be challenged as well, the auditor will perform the audit described in 3.2 using the locally generated round summary from step 2 and its copy of the witness values.

If the computed witness and stored witness values match, it is clear the object can be considered intact with a high probability. As the witness value is tied to a specific 24-hour window, the auditor is also able to assert an object existed in its current form since the supplied Integrity Token was issued.

# 4. IMPLEMENTATION AND WORKFLOW

The two software components that implement the audits described above are the ACE Audit Manager and the ACE Integrity Management Service. The Audit Manager component is installed and run locally at an archive to monitor their data. This AM provides an easy to use web interface for managing multiple collections stored on a variety of storage resources. The remote Integrity Management Service is installed at the University of Maryland and is publically available to issue Integrity Tokens.

While the core functionality of ACE is to secure cryptographic digests using Integrity Tokens and witness values, the distributed Audit Manager expands upon this service to provide a general integrity monitoring system for an archive.

## 4.1 Integrity Management Service

The Integrity Management Service installed at UMD provides the IMS functionality described above through a set of simple web services. These services are available at http://ims.umiacs.umd.edu:8080/ace-ims/IMSWebService and described through a Web Service Definition Language[8] document. The core services are described below:

1. Request Tokens Asynchronously - Request a set of tokens. This function will return immediately, requiring a client to perform a callback after the round timeout to request generated tokens.

2. Retrieve Tokens – Retrieve any previously requested tokens. This will return an error if a round has not yet completed.
3. Request Tokens Immediately – A blocking function that will wait until the current round completes and return any tokens requested tokens to the caller. As this may be require several seconds, clients that are bulk registering tokens should use the callback method described above.
4. Get Round Summaries – Get a list of round summaries for previously generated rounds.
5. Generate Proof For Round – Generate a proof linked the requested round to the daily witness value. Results for this are not available until after the daily witness value has been generated.

The IMS at umiacs is currently configured to generate witness values at 12:00am EST. These witness values are published to two publicly available mailing lists, one at UMD a second Google group. Anyone is free to subscribe to one or both of these lists to receive nightly notification of certificate generation.

## 4.2 Audit Manager (AM)

The Audit Manager is a java based web application which runs in the Tomcat[9] environment and uses a MySQL[10] database for its data storage. There have been several released of the AM with the current 1.4.3 having significantly expanded the role of the AM beyond just integrity token and object validation.

An AM handles both registration of new items and monitoring of existing items. The AM is able to request tokens for new items in collections, validate items against their stored digests, and verify those digests using integrity tokens and the IMS. The AM will not take corrective action when it encounters an error. That is beyond the scope of the software and better handled by infrastructure designed to identify complete replica copies.[2][7] Each collection is able to specify a different audit policy. It also provides complete logging of all actions performed against a collection as well as extensive browsing and reporting capability.

1. Multiple Collections – Multiple collections on different resources are able to be monitored through a single instance of an AM.
2. Detailed Logging – Every change observed during an audit is logged.
3. Reporting – Summary reports are generated and distributed both at the end of each audit round and upon a configurable schedule.
4. Customizable Policy – Collections may have independent policies regarding the frequency when they are audited.
5. Simple Administration – After the initial installation of the AM, all management is handled through the web interface. This allows an archivist or librarian to completely control audit policy without requiring the assistance of additional IT support staff.

# 5. DEPLOYMENT

ACE has been deployed for almost two years in the Chronopolis Preservation Environment[2]. In addition, the IMS has been serving tokens for over two years. This has allowed us to observe how ACE functions in a production environment. Some of these new features are described in tne next section.

During this time the IMS has performed over 2.18 million aggregation rounds and generated over 500 witness values. Total storage space on the IMS under 270MB. Publicly available copies of every witness value are still available on two different mail list archives.

The Chronopolis Preservation Environment is a consortium of three data storage partners, and several data providers. Data storage partners include the University of Maryland(UMD), San Diego Supercomputing Center(SDSC)/UCSD Libraries, and the National Center for Atmospheric Research(NCAR).

Currently, Chronopolis uses the Storage Resource Broker (SRB)[6] to provide data grid services. The SRB provides a unified namespace and common functionality for data access and placement while abstracting access to underlying data resources. Each Chronopolis partner has an SRB installation backed by a metadata catalog(MCAT) and disk storage. These SRB installations are federated, so each site is able to see shared data at other partner sites.

Collections in Chronopolis are provided by a set of data providers. Data first staged into the SRB installation at SDSC, then periodically replicated to resources at UMD and NCAR. Collections in Chronopolis vary both in the number of files and size of those files.

**Table 1. UMD Chronopolis Collections**

| Provider | Files | Directories | Size(GB) |
|---|---|---|---|
| CDL | 46,762 | 28 | 4,291 |
| SIO-GDC | 197,718 | 5,230 | 815 |
| ICPSR | 4,830,625 | 95,580 | 6,957 |
| NC-State | 608,424 | 42,207 | 5,465 |

The Chronopolis installation required an Audit Manager to be installed at each of the three partners. This allows each partner to define how its own policy for monitoring collections. Due to the nature of the three sites, the monitoring period differs between each. At UMD and NCAR collections are scanned every 30 days. SDSC is currently manually controlling audit start due to a requirement that data be staged to disk prior to auditing.

Over the past year, the UMIACS Chronopolis node has accumulated 12.2 million log events while monitoring 5.9 million files. Complete auditing of all collections at UMIACS takes roughly a week. The following table shows some observed performance against various chronopolis collections.

**Table 2. Audit Throuput**

| Provider | Time(h) | Files/s | Bandwidth(MB/s) |
|---|---|---|---|
| CDL | 20:32 | .63 | 59.44 |
| SIO-GDC | 6:49 | 8.05 | 34.00 |
| ICPSR | 122:48 | 10.93 | 16.11 |
| NC-State | 32:14 | 5.24 | 48.22 |

## 6. LESSONS LEARNED

During the past year, several revisions of the AM have been released in response to the needs of Chronopolis. The core Integrity token validation has remained unchanged, however as ACE is continually reading the files and maintaining digests on all monitored items, several additional features were easily added without expanding the scope of the AM. The most important of these are listed below.

1. Audit Throttling – Initially, ACE would process data at the speed an archive could supply. This sped up audits, however it caused some unanticipated load issues on the MCAT component in the sites SRB installations. Limits were added to control how many queries per second may be sent to an archive, as well as limits on how fast to transfer data from an archive. These throttles have proven effective in limiting the impact of large scale auditing on network and database resources.
2. Digest comparison – The AM has been expanded to peer with other Audit Managers to allow for the comparison of digest lists between different installations.
3. JSON output – Most core functionality of the AM is able to produce both html and JavaScript Object Notation[5] output. This allows 3rd party portals to be developed that aggregate the data from serveral AM installations.
4. Expanded Digest support – MD5 and SHA-256 digests are now supported to allow the digests calculated in the AM to be compared with existing digest lists generated by software such as BagIT.

The UMIACS Chronopolis node is configured to throttle SRB access by limiting the number of simultaneous audits to 3 and limiting queries to 10 per second. Through trial and error, these limits provided the best tradeoff between rapid collection auditing and minimizing the impact on the underlying archive. Table 2 shows collection auditing prior to these throttles. The only collection audit impacted by the query speed was the ICPSR audit. Prior to tuning, the MCAT would respond noticeably slower during an audit of this collection.

## 7. FUTURE DIRECTIONS

While the current ACE Audit Manager has shown that is can reliably monitor many terabytes of disk-based assets, other types of near-line and offline storage may require different auditing techniques. In the case of data stored in Chronopolis at SDSC, their hierarchical storage management system requires that data be loaded from tape prior to access. Given this type of access, ACE should be aware of data placement in the offline resources so that is can request access in an optimal fashion. This would require the AM to be modified to support a configurable set of access patterns that would be stored. Defining access patterns for data in the audit manager may even allow performance improvements when accessing large collections of files stored on disk resources.

Another area to expand ACE is into monitoring data stored in a cloud computing environment. Currently the ACE Audit Manager is designed to operate close to the data it is monitoring. In most clouds, there is a cost associated with transferring data into or out of cloud[3][4], therefore remotely running an Audit Manager to monitor cloud holdings is not feasible. However, ACE integrity tokens can still be used to secure cloud holdings. As the integrity tokens can be stored in an untrusted environment, they may be stored alongside data within a cloud. Verifying that integrity tokens have not been modified requires very little network access, therefore it is possibly to validate tokens within a cloud without incurring a high cost. These tokens may even be generated prior to inserting data into the cloud and incorporated into part of an organizations cloud ingestion process. Storing tokens in the cloud would allow applications running in a cloud to assert the data they are accessing has not been tampered with or damaged.

## 8. CONCLUSION

Over the past year, the ACE environment has shown that it can scale up the workloads required in a large scale archive. It has provided an easy to use interface for managing terabytes of data and millions of files. While ACE has proven it performs remarkably well for monitoring local data collections, we believe the ACE method for file authentication is well suited for managing data stored remotely in untrusted cloud resources.

## 9. REFERENCES


[1] Song S., JaJa J.: Techniques to audit and certify the long-term integrity of digital archives. In: International Journal on Digital Libraries, Volume 10, Numbers 2-3 / August, 2009 pp. 123-131

[2] Minor, D., D. Sutton, A. Kozbial, M. Burek, M. Smorul. Chronopolis Digital Preservation Network . To be published in The International Journal of Digital Curation, June 2010.

[3] Amazon Simple Storage Service http://aws.amazon.com/s3/

[4] Rackspace Cloudloud http://www.rackspacecloud.com

[5] Crockford, D.: The application/json Media Type for JavaScript Object Notation (JSON): http://www.ietf.org/rfc/rfc4627.txt

[6] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In Procs. of CASCON'98, Toronto, Canada, 1998

[7] Maniatis, P., Roussopoulos, M., Giuli, T.J., Rosenthal, D.S.H.,Baker, M.: The LOCKSS peer-to-peer digital preservation system. ACM Trans. Comput. Syst. 23(1), 2–50 (2005). http://doi.acm.org/10.1145/1047915.1047917

[8] Christensen, E, Curbera F., Meredith G., Weerawarana S.: Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001. http://www.w3.org/TR/wsdl

[9] Apache Tomcat. http://tomcat.apache.org

[10] MySQL. http://www.mysql.com